Journal of Computer Science and Cybernetics, V.41, N.1 (2025), 17–32
DOI no. 10.15625/1813-9663/21653

PARALLELIZATION OF THE FUNNEL TREE ALGORITHM FOR FINDING SHORTEST PATHS ON POLYHEDRAL SURFACES

PHAN THANH AN[1,2], TRINH MINH DUC[3,*], TRAN VAN HOAI[2,4]

[1]*Institute of Mathematical and Computational Sciences, Ho Chi Minh City University of Technology, 268 Ly Thuong Kiet, Ward 14, District 10, Ho Chi Minh City, Viet Nam*
[2]*Vietnam National University Ho Chi Minh City, 268 Ly Thuong Kiet, Ward 14, District 10, Ho Chi Minh City, Vietnam*
[3]*Faculty of Information Technology, Thai Nguyen University of Information & Communication Technology, Z115 Street, Quyet Thang Commune, Thai Nguyen City, Thai Nguyen Province, Viet Nam*
[4]*Faculty of Computer Science & Engineering, Ho Chi Minh City University of Technology, 268 Ly Thuong Kiet, Ward 14, District 10, Ho Chi Minh City, Vietnam*

Abstract. This paper presents an enhanced approach for computing shortest paths on triangulated polyhedral surfaces by parallelizing the Funnel Tree Algorithm (introduced by An et al. in The funnel tree algorithm for finding shortest paths on polyhedral surfaces, *Optimization* (2023)) and incorporating the Method of Orienting Curves. In particular, we use the Method of Orienting Curves for finding the shortest path from the cusp of a funnel to its direct destination. As a result, the children of the funnel are also determined simultaneously. This combined approach leverages modern multi-core processors to achieve significant performance improvements. Experimental results demonstrate the effectiveness of this method on various polyhedral surfaces. The resulting implementation achieves significantly better speedups over the corresponding sequential code given by An et al. when compared to their previous work.

Keywords. Funnel, shortest path, parallel method, method of orienting curves.

1. INTRODUCTION

Computing the shortest paths connecting two points on a polyhedral surface is an extensively researched computational geometry problem with several applications in fields including navigation, robotics, and GIS [1, 2].

Several researchers approached the problem of determining the shortest path connecting two points in a sequence of triangles of some polyhedral surface (the restricted shortest path problem, in short) as a solution to this problem, e.g. [3, 4]. Usually, the planar unfolding technique is applied to address this problem. Pham-Trong et al. [3] proposed an algorithm that computes the shortest path connecting two points in a sequence of triangles in detail. All of the sequence's triangles are unfolded into a common plane. After such planar unfolding,

*Corresponding author.
E-mail addresses: thanhan@hcmut.edu.vn (P.T. An); tmduc@ictu.edu.vn (T.M. Duc); hoait@hcmut.edu.vn (T.V. Hoai).

they took into consideration the planar unfolded sequence and used beam propagation to determine the shortest path. Using the same planar unfolding technique, Xin and Wang (2007) [4] described a planar unfolding process of a sequence of triangles and subsequently presented an algorithm for solving the restricted shortest path problem.

The Method of Orienting Curves was first presented by Phu [5] in 1987 as a solution for some optimal control problems with state constraints. Then it was developed further in [6, 7, 8, 9]. In particular, Steiner's problem of finding the polygon with minimal circumference that is inscribed in a (arbitrary) given convex polygon was solved with the help of the Method of Orienting Curves [10], requiring only a ruler and compass for construction. Furthermore, the optimal control problem of hydroelectric power plants [11], Zermelo's navigation problem along a river [12], the optimal inventory problem [8], and time-optimal control of manipulator along a prescribed trajectory [13] were the important practical problems that this method was successfully applied to solve.

An discovered recently that the idea of the Method of Orienting Curves can be effectively used to solve various computational geometry problems [14, 15, 16]. The shortest path connecting two points in the sequence of triangles was found by cusps of funnels associated with common edges along the sequence of triangles in [17], following the introduction of the concept of a funnel associated with a common edge along the sequence of triangles in three-dimensional space (which is comparable to Lee and Preparata's one in a simple polygon [18]). The Method of Orienting Curves was used to construct such a funnel without planar unfolding technique.

In [19], An et al. proposed an algorithm to build the funnel tree by recursively splitting funnels which operates in $O(n^2)$ time complexity, where n is the number of faces of the surface. This algorithm constructs a funnel tree to determine the shortest paths from a fixed source point to all other vertices on a triangulated polyhedral surface. While effective, the algorithm's sequential nature limits its scalability on large datasets and complex surfaces. With the advent of multi-core processors, there is a significant opportunity to improve the performance of computational geometry algorithms through parallelization. In this paper, we present a parallel algorithm to build a funnel tree based on the sequential algorithm in [19]. The resulting implementation achieves significantly better speedups over corresponding sequential code given in [19] when compared to a previous work. We also present the method of orienting curves for finding the shortest path from the cusp of a funnel to its direct destination. As a result, the children of the funnel are also determined simultaneously.

The rest of the paper is organized as follows. In Section 2, we introduce some definitions and properties. Then in Section 3, we propose an approach to determine children of a funnel by the Method of Orienting Curves. Section 4 presents our parallel algorithm. In Section 5, we present the improved parallel algorithm, and Section 6 then describes our implementation and experimental results. Section 7 concludes this paper.

2. PRELIMINARIES

Let us review certain definitions and properties. Let p and q be any two points in space, we denote $[p, q] := \{(1-\lambda)p+\lambda q : 0 \leq \lambda \leq 1\},]p, q] := [p, q]\backslash\{p\}$ and $[p, q]$ is called a line segment [17]. Several terminologies for three-dimensional space that were introduced in [2, 17, 20] will be used in this paper. A sequence of triangles S is defined by a list $(f_1, f_2, \ldots, f_{m+1})$ of

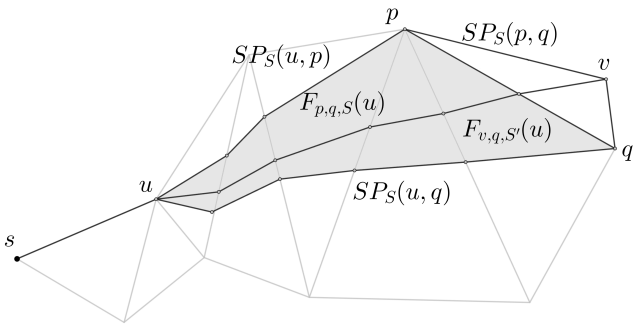

Figure 1: The sequence S, the funnel $F_{p,q,S}(u)$ (gray), its direct destination v and its child $F_{v,q,S'}$, where $S' = S \cup \triangle pqv$

adjacent triangles $f_1, f_2, \ldots, f_{m+1}$ on some triangulated polytope surface, where f_i and f_{i+1} share a common edge e_i. A list of common edges $E = (e_1, e_2, \ldots, e_m)$ is called a sequence of common edges of S. For points $p, q \in S$, denote by $SP_S(p, q)$ the shortest path joining p and q in S.

A path γ connecting two points p and q in S is a polyline $\bigcup_{i=0}^{n}[v_i, v_{i+1}]$, where $v_0 = p, v_{n+1} = q$, each line segment $[v_i, v_{i+1}]$ is on some triangle of S, each vertex $v_i (i \in 1, \ldots, n)$ of the path belongs to some common edge of S [21]. Denote the length of a polyline γ by $l(\gamma)$.

A path $\bigcup_{i=0}^{m}[v_i, v_{i+1}]$ in a sequence of triangles S, where $v_0 = u, v_{m+1} = v$, each line segment $[v_i, v_{i+1}]$ is on some triangle of S, each vertex v_i of the path belongs to some common edge of S, is a *straightest geodesic* from u to v in S if the measures of the angles of this path at all vertices $v_i (i \in 1, 2, \ldots, m)$ are $180°$ [17].

Such straightest geodesic is denoted by $\mathcal{C}_S(u, v)$. We also call a line segment on a triangle f_i a straightest geodesic [21].

Definition 1. ([19]) The region $F_{p,q,S}(u)$ is called a *funnel* (of the polytope) associated with the vertex u and the path $SP_S(p, q)$. The vertex u is called the *cusp* of the funnel, $SP_S(u, p)$ is called the *left border* and $SP_S(u, q)$ is called the *right border* of the funnel. If $[v, q]$ is an edge of the polytope, v is called the *direct destination* of the funnel $F_{p,q,S}(u)$.

Definition 2. ([19]) Consider a funnel $F_{p,q,S}$, its direct destination v and $S' = S \cup \triangle pqv$. If $F_{p,v,S'}$ or $F_{v,q,S'}$ exists (i.e. $F_{p,q,S}$ and $F_{p,v,S'}$ or $F_{v,q,S'}$ have the same cusp s), it is called a *child* of $F_{p,q,S}$.

Definition 3. ([19]) Assume that s is a vertex of the surface. A *funnel tree* is a tree with root s, where each node other than the root s is a triple $(F_{p,q,S}, SP_S(p, q), s)$, denoted by $F_{p,q,S}$. In the funnel tree, node $F_{p,v,S'}$ is a child of the node $F_{p,q,S}$ if the funnel $F_{p,v,S'}$ is a child of the funnel $F_{p,q,S}$.

The definitions of funnel and its direct destination are given in Figure 1.

3. DETERMINING CHILDREN OF A FUNNEL BY THE METHOD OF ORIENTING CURVES

3.1. Proposed method

The problem under consideration is as follows: given a sequence $f_1, f_2, \ldots, f_m$ of $m \geq 2$ adjacent triangles on the surface of a polytope. Let $S := \bigcup_{i=1}^{m} f_i$. For $i \in \{1, 2, \ldots, m-1\}$, f_i is a face, $e_i = f_i \cap f_{i+1}$ is an edge of the polytope. Consider a funnel $F_{p,q,S}$, its direct destination v and $F' = F \cup \triangle pqv$, s is the cusp of the funnel, F' is called the *processed region* of F. The question is how to find the shortest path $SP_{F'}(s, v)$ connecting two points s and v in the region F'.

In [19] An et al. constructed a funnel tree that contains the shortest paths from a source point s to all vertices of the surface, where the left borders of these funnels are straightest geodesics. We use the denotation $\mathcal{C}_{F'}(s, v)$ for a straightest geodesic from s to v in F'.

Based on the geometrical idea of [5], An [22] presents the Method of Orienting Curves in computational geometry (or in shorter notation denoted as MOC) for solving some geodesic problems in two-dimensional and three-dimensional spaces. The method has the following factors:

($e1$) determine the boundary for the solution,
($e2$) determine an initial element of the solution,
($e3$) construct the final curve and orienting curves through a given point,
($e4$) construct the restricted area from the boundary,
($e5$) the exact solution is constructed from a final curve and orienting curves.

In this paper we propose the Method of Orienting Curves for finding the shortest path from the cusp of a funnel to its direct destination in given F'. The method consists of the concepts of orienting curves and a final curve in a processed region (see Section 5 in [17] for more details). These concepts are straightest geodesics in a sequence of triangles. In the following definitions, the concepts of orienting and final curves of F' are presented for finding the shortest path $SP_{F'}(s, v)$ connecting two points s and v in F'.

Definition 4. ([17]) Let F' be the processed region, v be the direct destination of F. Set $B_1^{\text{ex}} := SP(s, p) \cup [p, v]$ and $B_2^{\text{ex}} := SP(s, q) \cup [q, v]$. Let z be a point in F.

1. If there exists a straightest geodesic in F' from z to v, then it is called a *final curve* of F' from the initial point z, and is denoted by $\text{FC}(z)$.
2. If there exists a longest straightest geodesic in F' from z, which ends at some point $q^* \in B_j^{\text{ex}}$, for some $j \in \{1, 2\}$, and meets B_i^{ex} at some point p^*, where $i \in \{1, 2\} \setminus \{j\}$, then it is called an *orienting curve* of F' from the initial point z and denoted by $\text{OC}(z)$. In addition, if p^* is the last point of $\text{OC}(z)$ on B_i^{ex}, then p^* is called a *transfer point* of the orienting curve $\text{OC}(z)$.

In Figure 2a, the straightest geodesic $\text{OC}(z)$ (the bold path connecting z and q^*) is an orienting curve of F' from the initial point z. In Figure 2b, $\mathcal{C}_{F'}(z, w_1)$ and $\mathcal{C}_{F'}(z, w_2)$ are not orienting curves of F' then $\mathcal{C}_{F'}(z, v)$ is a final curve of F' from z.

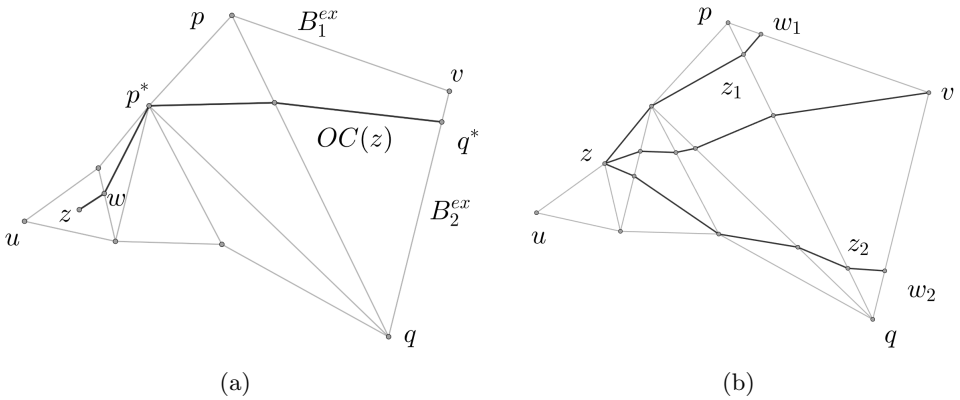

Figure 2: (a) OC(z) is an orienting curve of F' from the initial point z. (b) $\mathcal{C}_{F'}(z, v)$ is a final curve of F' from z

According to Theorem 5.1 in [19], we have exactly one final curve or one orienting curve of F' from z. If $z \in SP_{F'}(s, v)$ and FC(z) is the final curve of F' from the point z then FC(z) $= \mathcal{C}_{F'}(z, v)$ is a part of the shortest path $SP_{F'}(s, v)$. If $z \in SP_{F'}(s, v)$ and OC(z) is an orienting curve of F' from the initial point z with the transfer point z', then $z' \in SP_{F'}(s, v)$ and $\mathcal{C}_{F'}(z, z') \subset$ OC(z) is a part of the shortest path $SP_{F'}(s, v)$.

Next to determine children of a funnel, we have a remark as follows:

Remark 1. *The children of a funnel can be determined by finding the shortest path from the cusp of the funnel to its direct destination in given F'.*

This remark gives us an approach to determine children of a funnel, it differs from the one in [19] (where children of a funnel are determined by the law of cosines). Given a funnel, it was known that one of its borders is a straightest geodesic. Hence, when we use the method of orienting curves for finding the shortest path in [19], we just need to consider whether that straightest geodesic is an orienting curve or not. If not, we just need to go in the remaining direction to find the shortest path. The advantage here compared to [17] is that instead of having to go in two directions, we only need to go in one direction to reach the direct destination.

3.2. Numerical example

Let us consider a polytope with nine vertices, numbered 0 through 8. As shown in Figure 3, the polytope is triangulated. Let $S_1 = \triangle(4, 7, 5) \cup \triangle(7, 5, 6) \cup \triangle(6, 5, 2) \cup \triangle(2, 5, 1)$, $S_1' = S_1 \cup \triangle(2, 1, 8)$.

The source point is selected to be the vertex 4 ($s = 4$). The vertex 4 has 4 faces adjacent to it (see Figure 3). Coordinates of vertices of this polytope: 0(0,0,1); 1(0,1,1); 2(-1,1,1); 3(-1,0,1); 4(0,0,0); 5(0,1,0); 6(-1,1,0); 7(-1,0,0); 8(-2.24,1,3.24).

Consider the funnel $F_{1,2,S_1}(s)$ and the region $F' = F_{1,2,S_1}(s) \cap \triangle 128$, 8 is a direct destination of $F_{1,2,S_1}(s)$. We now use the orienting and final curves of F' for constructing the shortest path $SP_{F'}(s, 8)$ connecting two vertices $s = 4$ and 8 in F' and also determining

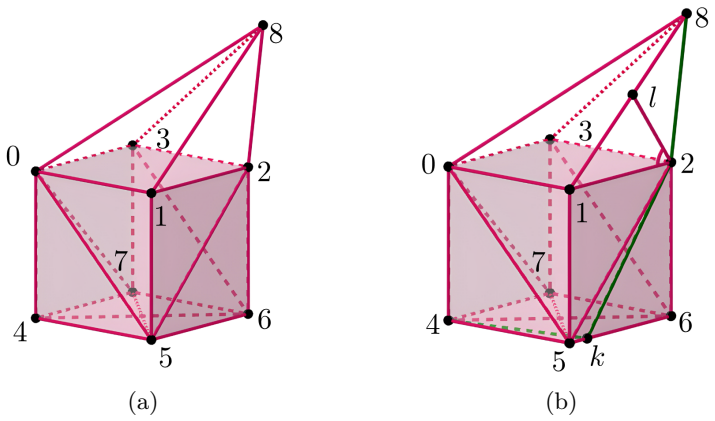

Figure 3: For s = 4, $S_1 = \triangle(4,7,5) \cup \triangle(7,5,6) \cup \triangle(6,5,2) \cup \triangle(2,5,1)$. The funnel $F_{1,2,S_1}(s)$. The polyline corlored in green is the shortest path $SP_{F'}(4,8)$ connecting two vertives 4 and 8.

children of F. In Figure 3, vertex 4 is the cusp of $F_{1,2,S_1}(s)$ associated with the common edge [1,2]. Consider $\triangle 128$, we have $\angle 812 \approx 45.17°, \angle 128 \approx 118.92°$.

Let $B_1(s)$ be the left border of $F_{1,2,S_1}(s)$, $B_1(s)$ passes through the vertices $4, 5, 1$. We see that $B_1(s)$ is the shortest path connecting 4 and 1 in $F_{1,2,S_1}(s)$. In addition, $B_1(s)$ is also a straightest geodesic from 4 to 1 in F. However, there does not exist a longest straightest geodesic from s having the same direction with $B_1(s)$ in F'. As a result, there does not exist an orienting curve of F' having the same direction with $B_1(s)$. Therefore, we need to go in the direction of the remaining border of $F_{1,2,S_1}(s)$. In Figure 3, to find the shortest path $SP_{F'}(s,8)$ in F', we call procedure NEWFUNNEL(F,8,2,4) (see [17] for more details). By the method of orienting curves, the shortest path connecting 4 to 8 in F' consists of parts of orienting curves and a final curve. The longest straightest geodesic starting at 4 and whose direction is the straightest geodesic from 4 to 2 in F' intersects $B_1^{ex} = B_1 \cup [1,8]$ at a point on $[1,8]$. Therefore, this longest straightest geodesic is an orienting curve of F' and 2 is its transfer point.

To find this orienting curve, we have to compute the angles $\angle 4k6$ and $\angle 6k2$. We have $\angle 46k = 45°$, $\angle 64k$ is computed by the formula (1) in [17]. Then, we have $\angle 64k \approx 35.264°$, $\angle 4k6 \approx 99.736°$. In Figure 3b, in order to the path γ joining the vertices $4, k, 2, l$ is a longest straightest geodesic in F', then $\angle 4k6 + \angle 2k6 = 180°$ and $\angle 12k + \angle 12l = 180°$. We get $\angle 2k6 \approx 80.264°$ and $\angle 12l \approx 99.736°$. The path γ is an orienting curve of F' and 2 is its transfer point.

The final curve FC(2) is a straighest geodesic from 2 to 8. It follows that the shortest path connecting 4 to 8 in F' is $\mathcal{C}_{F'}(4,2) \cup \mathcal{C}_{F'}(2,8)$.

After finding the shortest path $SP_{F'}(4,8)$, we can determine children of $F_{1,2,S_1}(s)$. In Figure 3b, the funnel $F_{1,2,S_1}(s)$ has one child $F_{1,8,S_1'}(s)$ which has two borders $SP_{F'}(4,1)$ and $SP_{F'}(4,8)$. $F_{8,2,S_1'}(s)$ is not a child of $F_{1,2,S_1}(s)$ because $SP_{F'}(4,8)$ intersects $SP_{F'}(4,2)$ at 2. Consequently, $F_{8,2,S_1'}(s)$ degenerates into a line segment [2,8] and its cusp is 2. According to Definition 3.2 in [19] this funnel is not a child of $F_{1,2,S_1}(s)$.

In [19], in order to find children of a given funnel, we use the law of cosines. Performing

the calculations we obtain similar results as above. The funnel $F_{1,2,S_1}(s)$ has one child $F_{1,8,S_1'}(s)$.

4. PARALLELIZING THE FUNNEL TREE ALGORITHM

4.1. Parallel algorithm

The divide-and-conquer strategy (see [23]) is used to give a parallelism of the sequential algorithm in [19]. To parallelize the Funnel Tree Algorithm, we divide the work among multiple processors, each processing a subset of funnels independently.

Algorithm 1 Parallel Algorithm

Input: s is a vertex of the polyhedral surface.
Output: A funnel tree that contains the shortest paths from s to all vertices of the surface.

1: $root := s$
2: **for** each edge $[p, q]$ opposite to s **do**
3: Insert $F_{p,q,S}$ as root's children
4: **end for**
5: **for** $k = 1, 2, \ldots, n$ **do** ▷ n is the number of faces
6: **Parallel for** each funnel (node) $F_{p,q,S}$ at the k^{th} level **do**
7: Find direct destination v of $F_{p,q,S}$ such that $\beta_v < 180°$ ▷ β_v is computed by the formula (4) in [19]
8: **if** such vertex does not exist **then**
9: $F_{p,q,S}$ has no children, **Continue**
10: **else**
11: Let $S' := S \cup \triangle pqv$
12: **if** $\angle psv < \angle psw$ **then**
13: $F_{p,q,S}$ has 2 children $F_{p,v,S'}, F_{v,q,S'}$. Insert them into the funnel tree and mark $\angle pvq$
14: **else**
15: $F_{p,q,S}$ has one child $F_{p,v,S'}$. Insert it into the funnel tree
16: **end if**
17: **end if**
18: **end for**
19: **for** each funnel $F_{p,q,S}$ at the k^{th} level **do**
20: **if** $\angle pvq$ is previously marked by another funnel called F_{p,q,S_1} **then**
21: Call CLIP_OFF_FUNNELS$(\triangle pqv, S, S_1)$ ▷ See procedure CLIP_OFF_FUNNELS in [19]
22: **end if**
23: **end for**
24: **if** $(k+1)^{th}$ level has no nodes **then**
25: **Break**
26: **end if**
27: **end for**

Synchronization mechanisms are used to ensure data consistency. At each level of the

funnel tree, we identify independent sections that can be processed concurrently. This involves partitioning funnels at each level into groups, ensuring that their processing does not interfere with each other. Each funnel will be processed by a separate processor.

We utilize a master-slave model, where one processor is designated as the master processor, while the remaining processors serve as slaves. The master processor is responsible for partitioning funnels at each level, distributing data to the slave processors and synchronizing. The slave processors process funnels in order to determine new funnels. However, in cases where the number of parallel tasks exceeds the number of slave processors, it is possible for a single slave processor to be assigned multiple tasks. The overall process is described as follows:

- The master processor partitions funnels at each level according to a number of funnels at each level. Let's assume that the number of funnels is denoted by n.
- If there are m slave processors and $n \leq m$, each node is assigned to one slave processor. Otherwise, each slave processor receives one funnel and the master processor monitors all the working processors. Next, repeatedly assign a funnel to the slave processor that finishes its task first.
- The master processor sends jobs which are funnels to slave processors at the $i-th$ level. After the slave processors have completed their tasks, they send results which are new funnels at the $(i+1)-th$ level to the master processor. The master processor synchronizes the results, then it it keeps sending funnels to slave processors at the $(i+1)-th$ level.

4.2. Correctness of the parallel algorithm

To prove the correctness of the parallel algorithm, we must show that the division into independent sections at each level does not affect the correctness of the computed shortest paths.

Proposition 4.1 *Given a triangulated polyhedral surface $\mathbb{P}$ with n vertices and a vertex s of $\mathbb{P}$, Algorithm 1 computes shortest paths from s to the other vertices of $\mathbb{P}$ by constructing a funnel tree in parallel using k processors, where $k \geq 2$. Proof.* There is no denying that the left border of the funnel $F_{p,q,S}$ forms a locally shortest path from s to p. A non-deleted child determines a locally shortest path from s to v. The initial set of funnels is created based on the source vertex s and the adjacent triangles. This step is performed sequentially, ensuring the correctness of the initial setup. At $k = 1$, the tree consists only of triangles incident to s, which represent the shortest paths from s to all vertices within those triangles. Assuming that the algorithm correctly identifies the shortest paths up to level k, at level $k+1$, the parallel algorithm ensures that the shortest path to any vertex is correctly computed by extending from the previously computed shortest paths. This follows from the sequential algorithm's properties, where each funnel's children are determined using geometric properties (angles and distances). The parallel execution simply distributes this work but does not alter the logic. The algorithm includes a procedure to "clip off" redundant funnels, ensuring that only the necessary funnels remain in the tree. This is crucial for both efficiency and correctness, as it prevents the exploration of unnecessary paths. In the parallel version, each processor clips its funnels independently, but the synchronization mechanism ensures that the global funnel

tree remains correct. The sequential algorithm in [19] uses geometric checks (e.g., angle comparisons) to determine whether a funnel has one or two children and to ensure that the shortest paths are updated correctly. This logic is preserved in the parallel algorithm, as each processor performs these checks independently but in parallel with others.

The parallel algorithm is correct because it preserves the core logic of the sequential version, ensuring that the shortest paths are computed correctly at each level of the tree. The parallelization simply distributes the workload but does not alter the algorithm's core guarantees. ■

In our parallel algorithm, after each funnel division, the master processor sends funnels to slave processors and waits for the slaves' calculations to complete before the master processor synthesizes and keeps sending new funnels for the next level. Therefore, if a slave processor returns slowly, the master processor also has to wait for it. This causes certain delays and increases the computational time. In the next section, we introduce an improvement of our parallel algorithm to further enhance the parallelization and reduce the computational time.

5. IMPROVED PARALLEL ALGORITHM

5.1. Improved parallel algorithm

Similar to Algorithm 1 in Section 4, the divide-and-conquer strategy is employed to introduce parallelism. At the first level of the funnel tree, independent sections are identified for concurrent processing. Funnels at the first level are partitioned into groups, ensuring their processing does not interfere with one another. Each group of funnels is then assigned to a separate processor for parallel execution.

If we have a lot of idle processors, we can repeat for subsequent levels by identifying child funnels at the 2^{nd} level, the 3^{rd} level, etc.: partition the child funnels of nodes at the 2^{nd} level or the 3^{rd} level into groups for parallel processing.

We employ a master-slave model, where one processor acts as the master, and the remaining processors function as slaves. The master processor handles partitioning the funnels at the 1^{st} level, distributing data to the slave processors, and managing synchronization. Slave processors are tasked with executing the procedure $\mathbf{FT}(F_j^*)$ to determine new funnels. If the number of parallel tasks exceeds the number of slave processors, a single slave processor may be assigned multiple tasks. The overall process is outlined as follows:

- The master processor partitions nodes at the 1^{st} level according to a number of nodes at the 1^{st} level. Let's assume that the number of nodes is denoted by n.

- If there are m slave processors and $n \leq m$, each node is assigned to one slave processor. Otherwise, each slave processor receives one node and the master processor monitors all the working processors. Next, repeatedly assign a node to the slave processor that finishes its task first.

- The master processor supervises the execution of each slave processor until all of slave processors have completed their tasks.

Algorithm 2 Improved parallel algorithm

Input: s is a vertex of the polyhedral surface.

Output: A funnel tree that contains the shortest paths from s to all vertices of the surface. We use k processors $0, 1, \ldots, k-1$ for the work ($k \geq 2$).

1: $root := s$
2: **for** each edge $[p,\ q]$ opposite to s **do**
3: Set $S = \triangle spq$
4: Insert $F_{p,q,S}$ as root's children.
5: **end for**
6: Let $F^* = \{f_1^*, f_2^*, \ldots, f_m^*\}$ be the set of root's children.
7: The set F^* is partitioned into $F_0^*, F_1^*, \ldots, F_{k-1}^*$. Each process $j (j = 0, 1, \ldots, k-1)$ calls **procedure FT**(F_j^*). ▷ **Procedure FT**(F_j^*) starts implementing from Step 5 of the sequential algorithm in [19]
8: **procedure FT**(F_j^*)
9: **while** $k \leq n$ and the k^{th} level has nodes **do**
10: Let $v = v_j$ = direct destination of the funnel $F_{p,q,S}$ and β_v be determined by (4) in [19].
11: **while** $\beta_v < \pi$ **do**
12: Take the sequence S' of adjacent triangles of the polytope between $[p, q]$ and $[q, v]$ having two edges incident at q.
13: Set $S' := S \cup \triangle pqv$.
14: **if** the funnels $F_{p,v,S'}(t_1)$ and $F_{v,q,S'}(t_2)$ have the same cusp s (i.e., $s = t_1 = t_2$) and (6) in [19] holds **then**
15: $F_{p,q,S}$ has two children $F_{p,v,S'}$, $F_{v,q,S'}$
16: Insert the children that $F_{p,q,S}$ can have as follows
17: **if** $\angle pvq$ is previously marked by another funnel called F_{p,q,S_1} **then**
18: Call procedure CLIP_OFF_FUNNELS$(\triangle pqv, S, S_1)$
19: Insert both children of $F_{p,q,S}$ into the funnel tree and mark $\angle pvq$
20: **else**
21: $F_{p,q,S}$ has one child, then insert the child into the funnel tree.
22: **end if**
23: **end if**
24: $k+ = 1$
25: **end while**
26: **end while**
27: **end procedure**

5.2. Correctness of the improved parallel algorithm

To prove the correctness of the improved parallel algorithm, we must show that the division into independent sections at the first level does not affect the correctness of the computed shortest paths.

Proposition 5.1 *Given a triangulated polyhedral surface $\mathbb{P}$ with n vertices and a vertex s of $\mathbb{P}$, Algorithm 2 computes shortest paths from s to the other vertices of $\mathbb{P}$ by constructing a funnel tree in parallel using k processors, where $k \geq 2$.*

Proof. Similarly to Proposition 4.1, the improved parallel algorithm is correct because it preserves the core logic of the sequential algorithm, ensuring that the shortest paths are computed correctly at each level of the tree. The parallelization simply distributes the workload but does not alter the algorithm's core guarantees. The initial set of funnels is created based on the source vertex s and the adjacent triangles. This step is performed sequentially, ensuring the correctness of the initial setup.

At the 1^{st} level, the master processor partitions the funnels into independent groups. Each group is processed in parallel by a slave processor by calling procedure $\mathbf{FT}(F_j^*)$. After slave processors have completed their tasks, they will send the branchs of the funnel tree to the master processor, this ensures that the shortest paths within each group are correctly computed.

The results from each group are merged to form the complete funnel tree at each level. Since the groups are independent, their merging does not affect the correctness of the shortest paths.

By induction, if the algorithm correctly computes the shortest paths up to a certain level using independent sections, then the correctness is maintained at the next level when sections are processed in parallel.

Therefore, Algorithm 2 maintains the correctness of the shortest path computations. ■

6. IMPLEMENTATION AND EXPERIMENTAL RESULTS

6.1. Setup

To evaluate the performance of our parallel algorithm, we conducted a series of experiments comparing the execution times of the original and parallelized versions. The experimental setup includes:

- **Hardware**: A multi-core processor Intel core i7 with 8 cores, providing sufficient computational power to test the parallelization.
- **Software**: C++ with OpenMP, enabling easy implementation and testing of parallel algorithms.
- **Datasets**: Various polyhedral surfaces with different numbers of vertices and faces, including both simple shapes like cubes and more complex shapes like spheres.

We used two primary metrics to evaluate the performance of the algorithms:

- **Execution time**: The total mean time taken to compute the shortest paths for a given dataset. This metric provides a direct measure of the algorithm's efficiency. For each dataset, we run algorithms 100 times, then we compute mean times and variances.
- **Speedup**: The ratio of the execution time of the sequential algorithm in [19] to our parallel algorithms. Speedup indicates how much faster the parallel algorithm is compared to the sequential algorithm in [19], with higher values indicating better performance.

6.2. Results and analysis

The experimental results in Tables 1 and 2 show that both our parallel and improved parallel algorithms significantly reduce execution time compared to the sequential algorithm in [19], achieving speedups of up to 1.4535 and 3.1902 times, respectively. These improvements come from better workload distribution across processors, enabling simultaneous processing of different parts of the funnel tree. The speedup is close to ideal linear performance, showing efficient resource use. Additionally, Algorithm 2 runs faster than Algorithm 1 due to less waiting time for processors, reducing delays and overall computational time.

Next, we analyze the experimental results in more detail, focusing on the differences in average execution time, variance, and relative variance (Coefficient of Variation, CV) between the sequential algorithm in [19] and our improved parallel algorithm. This analysis will highlight any imbalances and provide insight into the efficiency of the improved parallel algorithm. A summary of the average execution time and variance for each dataset in both algorithms is presented in Table 3 and Table 4.

Table 1: The actual execution times of the sequential algorithm in [19] and our parallel algorithm (time in milliseconds) with $k = 12$ processes.

Dataset	Vertices	Nodes	Sequential algorithm	Parallel algorithm	Speedup
Cube1	98	2016	12.8708	12.5859	1.0226
Cube2	200	4333	42.2883	33.4357	1.2648
Cube3	248	5419	56.7012	50.3575	1.1260
Cube4	296	6506	74.1305	65.5461	1.1310
Sphere1	200	3123	18.3007	17.1518	1.0670
Sphere2	400	6370	50.3543	42.0272	1.1981
Sphere3	600	10028	97.7762	68.7293	1.4226
Sphere4	1050	17608	262.6000	180.6670	1.4535
Spiral1	150	2477	14.9250	11.6283	1.2835
Spiral2	350	5788	76.5948	56.3611	1.3590

Table 2: The actual execution times of the sequential algorithm in [19] and our improved parallel algorithm (time in milliseconds) with $k = 12$ processes.

Dataset	Vertices	Nodes	Sequential algorithm	Improved parallel algorithm	Speedup
Cube1	98	2016	12.8708	6.1941	2.0779
Cube2	200	4333	42.2883	14.9967	2.8198
Cube3	248	5419	56.7012	19.2585	2.9442
Cube4	296	6506	74.1305	24.3539	3.0439
Sphere1	200	3123	18.3007	8.0578	2.2711
Sphere2	400	6370	50.3543	18.1875	2.7686
Sphere3	600	10028	97.7762	31.8382	3.0710
Sphere4	1050	17608	262.6000	82.3141	3.1902
Spiral1	150	2477	14.9250	6.8679	2.1731
Spiral2	350	5788	76.5948	28.9496	2.6458

The speedup indicates how much faster the parallel algorithm is compared to the sequential one. As shown in Tables 1 and 2, the highest speedup is for the Sphere4 dataset, achieving 3.1902 times, demonstrating the improved parallel algorithm's efficiency with larger datasets. Medium-sized datasets like Cube2 and Sphere2 show speedups between 1.1 and 2.9 times, benefiting from parallelism but less so than larger datasets. The smallest speedup, 1.0226 times, is seen with Cube1, suggesting that parallel overhead reduces effectiveness for smaller datasets.

Variance reflects the stability and consistency of the algorithm's execution times across runs. Lower variance indicates more predictable performance, while higher variance suggests fluctuations. For most datasets, parallel execution reduces variance, indicating greater stability. For instance, Sphere2's variance drops from 4.4483 in the sequential version to 0.7739 in parallel. However, some datasets, like Spiral2, show increased variance in parallel execution, possibly due to resource contention or non-deterministic task scheduling.

Table 3: Relative variances of the sequential algorithm in [19]

Dataset	Sequential algorithm	Variance	Standard deviation (σ)	Relative Variance CV
Cube1	12.8708	1.7732	1.3316	0.1035
Cube2	42.2883	7.5349	2.7450	0.0649
Cube3	56.7012	6.9831	2.6426	0.0466
Cube4	74.1305	11.4808	3.3883	0.0457
Sphere1	18.3007	1.2416	1.1143	0.0609
Sphere2	50.3543	4.4483	2.1091	0.0419
Sphere3	97.7762	11.3833	3.3739	0.0345
Sphere4	262.6000	29.7394	5.4534	0.0208
Spiral1	14.9250	0.1265	0.3557	0.0238
Spiral2	76.5948	3.1848	1.7846	0.0233

Table 4: Relative variances of our improved parallel algorithm

Dataset	Improved parallel algorithm	Variance	Standard deviation (σ)	Relative Variance CV
Cube1	6.1941	1.0957	1.0468	0.1690
Cube2	14.9967	2.4524	1.5660	0.1044
Cube3	19.2585	5.3482	2.3126	0.1201
Cube4	24.3539	4.1120	2.0278	0.0833
Sphere1	8.0578	0.2471	0.4971	0.0617
Sphere2	18.1875	0.7739	0.8797	0.0484
Sphere3	31.8382	5.0683	2.2513	0.0707
Sphere4	82.3141	12.7663	3.5730	0.0434
Spiral1	6.8679	0.1503	0.3877	0.0565
Spiral2	28.9496	5.7470	2.3972	0.0828

To evaluate relative variance across datasets, we use the coefficient of variation (CV), which normalizes variance by the mean, providing a clearer picture of fluctuations relative

to average execution time. As seen in Tables 3 and 4, CV values are low for most datasets, reflecting stable performance in sequential execution. However, Cube1 has the highest CV (0.1690) in parallel execution, indicating notable instability, likely due to load balancing issues or overhead with smaller datasets. Cube3 also shows a higher CV (0.1201) in parallel execution, reflecting more inconsistency compared to sequential runs. In contrast, larger datasets like Sphere2 and Sphere4 exhibit low CV values, showing that parallel execution is both fast and consistent.

As the datasets grow in size and complexity, the improved parallel algorithm tends to show a higher CV, although still generally more stable in the larger datasets. For example, Spiral2 has a CV of 0.0828 (parallel) compared to 0.0233 (sequential), indicating that the improved parallel algorithm is still performing relatively better in handling the larger datasets.

From this analysis, we can conclude that:

- Parallel execution is highly effective for large datasets such as **Sphere4**, **Sphere3** and **Cube4**, where the workload can be efficiently distributed across processors.
- Smaller datasets such as **Spiral1** suffer from increased variance and instability during parallel execution, with a significant increase in CV due to parallel overhead.
- Mid-sized datasets like **Spiral2** experience moderate speedup but show increased variance, suggesting the need for optimization in parallel execution.

7. CONCLUSION

The experimental results demonstrate the effectiveness of the parallelization strategy. By distributing work across multiple processes and processing independent sections, our parallel algorithms achieve significant speedups and improved performance. In summary, parallel execution is highly beneficial for large datasets, offering both speed and stability, whereas smaller datasets may face imbalances due to parallel overhead, leading to less consistent performance. Additionally, we propose the method of orienting curves for finding the shortest path from the cusp of a funnel to its direct destination in given F' (F' is called the *processed region* of the funnel F). This gives us a new approach to determine children of a funnel.

ACKNOWLEDGMENTS

The authors wish to express their thanks to Le Huu Trieu and Nguyen Van An (Faculty of Computer Science & Engineering, Ho Chi Minh City University of Technology, Ho Chi Minh City) for supporting this research.

This research is funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under Grant Number 102.01-2023.48.

REFERENCES

[1] R. Kimmel and J. A. Sethian, "Computing geodesic paths on manifolds," in *Proceedings of the National Academy of Sciences*, 1998.

[2] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou, "The discrete geodesic problem," *SIAM Journal on Computing*, vol. 16, no. 4, pp. 647–668, 1987.

[3] V. P. Trong, N. Szafran, and L. Biard, "Pseudo-geodesics on three-dimensional surfaces and pseudo-geodesic meshes," *Numerical Algorithms*, vol. 26, no. 4, pp. 305–315, 2001.

[4] S. Q. Xin and G. J. Wang, "Efficiently determining a locally shortest path on polyhedral surfaces," *Computer-Aided Design*, vol. 39, no. 12, pp. 1081–1090, 2007.

[5] H. X. Phu, "On the solution of a regular class of optimal control problems on a large scale using orientation curves (zur lösung einer regulären aufgabenklasse der optimalen steuerung im großen mittels orientierungskurven)," *Optimization*, vol. 18, no. 1, pp. 65–81, 1987.

[6] N. Dinh and H. X. Phu, "Solving a class of regular optimal control problems with state constraints by the method of orienting curves," *Optimization*, vol. 25, no. 2 & 3, pp. 231–247, 1992.

[7] ——, "Solving a class of optimal control problems which are linear in the control variable by the method of orienting curves," *Acta Math Vietnam*, vol. 17, no. 2, pp. 115–134, 1992.

[8] H. X. Phu, "Method of orienting curves for solving optimal control problems with state constraints," *Numerical Functional Analysis and Optimization*, vol. 12, no. 1 & 2, pp. 173–211, 1991.

[9] H. X. Phu and N. Dinh, "Some remarks on the method of orienting curves," *Numer Funct Anal Optim*, vol. 16, no. 5 & 6, pp. 755–763, 1995.

[10] H. X. Phu, "A constructive solution method for the problem of the polygon with the smallest perimeter by j. steiner (ein konstruktives lösungsverfahren für das problem des inpolygons kleinsten umfangs von j. steiner)," *Optimization*, vol. 18, no. 3, pp. 349–359, 1987.

[11] N. Dinh and H. X. Phu, "The method of orienting curves and its application to an optimal control problem of hydroelectric power plants," *Vietnam Journal of Mathematics*, vol. 20, pp. 40–53, 1992.

[12] H. X. Phu, "On the solution of a zermelo navigation problem (zur lösung eines zermeloschen navigationsproblems)," *Optimization*, vol. 18, pp. 225–236, 1987.

[13] H. X. Phu, H. G. Bock, and J. Schlöder, "The method of orienting curves and its application for manipulator trajectory planning," *Numerical Functional Analysis and Optimization*, vol. 18, pp. 213–225, 1997.

[14] P. T. An, "Method of orienting curves for determining the convex hull of a finite set of points in the plane," *Optimization*, vol. 59, pp. 175–179, 2010.

[15] P. T. An and L. H. Trang, "An efficient convex hull algorithm for finite point sets in 3D based on the method of orienting curves," *Optimization*, vol. 62, pp. 975–988, 2010.

[16] P. T. An, T. M. Duc, and D. T. Oanh, "OFC-DelaunayTriangulation: A new efficient algorithm for merging two adjacent delaunay triangulations," *Discrete Mathematics, Algorithms and Applications*, vol. 16, no. 2, pp. 975–988, 2010.

[17] P. T. An, "Finding shortest paths in a sequence of triangles in 3D by the method of orienting curves," *Optimization*, vol. 67, no. 1, pp. 159–177, 2018.

[18] D. T. Lee and F. P. Preparata, "Euclidean shortest paths in the presence of rectilinear battiers," *Networks*, vol. 14, no. 3, pp. 393–410, 1984.

[19] P. T. An, T. V. Hoai, and V. B. Thinh, "The funnel tree algorithm for finding shortest paths on polyhedral surfaces," *Optimization*, 2023.

[20] N. N. Hai, P. T. An, and P. T. T. Huyen, "Shortest paths along a sequence of line segments in euclidean spaces," *Journal of Convex Analysis*, vol. 26, no. 4, pp. 1089–1112, 2019.

[21] P. T. An, "Finding shortest paths in a sequence of triangles in 3D by the planar unfolding," *Numerical Functional Analysis and Optimization*, vol. 40, no. 8, pp. 944–952, 2019.

[22] ——, *Optimization Approaches for Computational Geometry.* Hanoi: Publishing House for Science and Technology, Vietnam Academy of Science and Technology, 2017.

[23] J. JáJá, *Introduction to Parallel Algorithms.* Addison-Wesley, 1992.

Received on November 17, 2024
Accepted on February 28, 2025